Java Chapter 7: Arrays and ArrayLists, and HashMaps

Contents

Time required: 60 minutes

Do: Think Java, 2nd Ed. (Interactive Edition)

- Chapter 7 Arrays

Do: Introduction to Programming with Java

- Chapter 8 ArrayLists

Do: Online Tutorial

Go through the following tutorials.

- Java Arrays
- Loop Through an Array
- Multidimensional Arrays
- Java ArrayLists
- Java HashMaps
- Java YouTube Tutorial - 01 - Declaring Arrays & Accessing Elements
- Chapter 6.6 http://itec2140.gitlab.io/#_arrays
- Chapter 6.7 http://itec2140.gitlab.io/#_arraylists

Arrays

We will learn how to store multiple values of the same type by using a single variable. This language feature will enable you to write programs that manipulate larger amounts of data.

An array is a sequence of values; the values in the array are called elements. You can make an array of ints, doubles, Strings, or any other type, but all the values in an array must have the same type.

To create an array, you declare a variable with an array type, then create the array itself. Array types look like other Java types, except they are followed by square brackets ([]). For example, the following lines declare that counts is an "integer array" and values is a "double array":

```
int[] counts;
double[] values;
```

To create the array itself, you use the **new** operator. The new operator allocates memory for the array and automatically initializes all its elements to zero:

```
counts = new int[4];
values = new double[size];
```

The first assignment makes counts refer to an array of four integers. The second makes values refer to an array of doubles, but the number of elements depends on the value of size (at the time the array is created).

You can also declare the variable and create the array with a single line of code:

```
int[] counts = new int[4];
double[] values = new double[size];
```

You can use any integer expression for the size of an array, if the value is nonnegative. If you try to create an array with -4 elements, for example, you will get a NegativeArraySizeException. An array with zero elements is allowed, and there are special uses for such arrays.

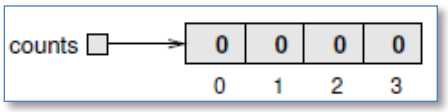
You can initialize an array with a comma-separated sequence of elements enclosed in braces, like this:

```
int[] a = {1, 2, 3, 4};
```

This statement creates an array variable, a, and makes it refer to an array with four elements.

Accessing Elements

When you create an array with the new operator, the elements are initialized to zero. The following diagram shows a memory diagram of the counts array.



The arrow indicates that the value of counts is a reference to the array. You should think of the array and the variable that refers to it as two different things. As you'll soon see, we can assign a different variable to refer to the same array, and we can change the value of counts to refer to a different array.

The boldface numbers inside the boxes are the elements of the array. The lighter numbers outside the boxes are the indexes used to identify each location in the array. As with strings, the index of the first element is 0, not 1.

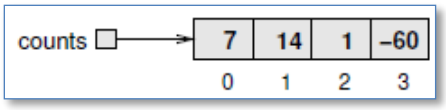
The [] operator selects elements from an array:

```
System.out.println("The first element is " + counts[0]);
```

You can use the [] operator anywhere in an expression:

```
counts[0] = 7;
counts[1] = counts[0] * 2; counts[2]++;
counts[3] -= 60;
```

The following diagram shows the results of these statements.



You can use any expression as an index if it has type int. One of the most common ways to index an array is with a loop variable. For example:

```
int i = 0;
while (i < 4){
    System.out.println(counts[i]);
    i++;
}
```

This while loop counts from 0 to 4. When i is 4, the condition fails, and the loop terminates. The body of the loop is executed only when i is 0, 1, 2, or 3. In this context, the variable name i is short for "index".

Each time through the loop, we use **i** as an index into the array, displaying the ith element. This type of array processing is usually written as a for loop:

```
for (int i = 0; i < 4; i++) {
    System.out.println(counts[i]);
}
```

For the counts array, the only legal indexes are 0, 1, 2, and 3. If the index is negative or greater than 3, the result is an ArrayIndexOutOfBoundsException.

Displaying Arrays

You can use println to display an array, but it probably doesn't do what you would like. For example, say you print an array like this:

```
int[] a  = {1, 2, 3, 4};
System.out.println(a);
```

The output is something like this:

```
[I@bf3f7e0
```

The bracket indicates that the value is an array, I stands for "integer", and the rest represents the address of the array in memory.

If we want to display the elements of the array, we use a loop to iterate through the data structure:

```
public static void printArray(int[] a) {
    for (int i = 1; i < a.length; i++){
        System.out.print(a[i]);
    }
}
```

Given the previous array, the output of printArray is as follows:

```
1 2 3 4
```

The Java library includes a class, java.util.Arrays, that provides methods for working with arrays. One of them, toString, returns a string representation of an array. After importing Arrays, we can invoke toString like this:

```
System.out.println(Arrays.toString(a));
```

The output is shown here.

```
[1, 2, 3, 4]
```

Traversing or Iterating Arrays

Many computations can be implemented by looping through the elements of an array and performing an operation on each element. Looping through the elements of an array is called traversing or iterating:

```
int[] anArray = {1, 2, 3, 4, 5};
for (int i = 0; i < anArray.length; i++){
    anArray[i] *= anArray[i];
}
```

This example traverses an array and squares each element. At the end of the loop, the array has the values {1, 4, 9, 16, 25}.

Another common pattern is a search, which involves traversing an array and "searching" for a particular element. For example, the following method takes an array and a value, and it returns the index where the value appears:

```
// Main program
public class ArraySearchMethod {
```

```
    public static void main(String[] args) {
        double[] anArray = { 1.0, 2.0, 3.0 };
        // Search an array using a method with parameters
        int index = search(anArray, 2.0);
        System.out.println("\nindex = " + index);
    }

    /**
     * Returns the index of the target in the array, or -1 if not found
     */
    public static int search(double[] array, double target) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == target) {
                return i;
            }
        }
        return -1; // not found
    }
}
```

If we find the target value in the array, we return its index immediately. If the loop exits without finding the target, it returns -1, a special value chosen to indicate a failed search. (This code is essentially what the String.indexOf method does.)

The following code searches an array for the value 1.23, which is the third element. Because array indexes start at 0, the output is 2:

```
double[] array = {3.14, -55.0, 1.23, -0.8};
int index = search(array, 1.23);
System.out.println(index);
```

Another common traversal is a reduce operation, which "reduces" an array of values down to a single value. This can also be considered to be a running total of the values in the array.

Examples include the sum or product of the elements, the minimum, and the maximum. The following method takes an array and returns the sum of its elements:

```
public static double sum(double[] array){
    double total = 0.0;
    for (int i = 0; i < array.length; i++){
        // Accumulate or sum the values in the array
        total += array[i];
```

```
    }
    return total;
}
```

Before the loop, we initialize total to 0. Each time through the loop, we update total by adding one element from the array. At the end of the loop, total contains the sum of the elements. A variable used this way is sometimes called an accumulator because it "accumulates" the running total.

Random Numbers

Most computer programs do the same thing every time they run; programs like that are called deterministic. Usually, determinism is a good thing, since we expect the same calculation to yield the same result. But for some applications, we want the computer to be unpredictable. Games are an obvious example, but there are many others, like scientific simulations.

Making a program nondeterministic turns out to be hard, because it's im- possible for a computer to generate truly random numbers. But there are algorithms that generate unpredictable sequences called pseudorandom numbers. For most applications, they are as good as random.

java.util.Random generates pseudorandom numbers. The method nextInt takes an integer argument, n, and returns a random integer between 0 and n - 1 (inclusive).

If you generate a long series of random numbers, every value should appear, at least approximately, the same number of times. One way to test this behavior of nextInt is to generate many values, store them in an array, and count the number of times each value occurs.

The following method creates an int array and fills it with random numbers between 0 and 99. The argument specifies the desired size of the array, and the return value is a reference to the new array:

```
public static int[] randomArray(int size){
    Random random = new Random();
    int[] a = new int[size];
    for (int i = 0; i < a.length; i++){
        a[i] = random.nextInt(100);
    }
    return a;
}
```

The following main method generates an array and displays it by using the printArray method from earlier:

```
import java.util.Random;

public class RandomArray {
    public static void main(String[] args) {
        int[] array = randomArray(8);
        printArray(array);
    }

    /**
     * Fills and returns an array of random integers
     */
    public static int[] randomArray(int size) {
        Random random = new Random();
        int[] a = new int[size];
        for (int i = 0; i < a.length; i++) {
            a[i] = random.nextInt(100);
        }
        return a;
    }

    /**
     * Prints the elements of an array
     */
    public static void printArray(int[] a) {
        // Print array up until the last element
        for (int i = 0; i < a.length - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        // Print the last element of the array
        System.out.println(a[a.length - 1]);
    }
}
```

Each time you run the program; you should get different values. The output will look something like this:

```
15, 62, 46, 74, 67, 52, 51, 10
```

The Enhanced for Loop

Since traversing arrays is so common, Java provides an alternative syntax that makes the code more compact. Consider a for loop that displays the elements of an array on separate lines:

```
for (int i = 0; i < values.length; i++){
    int value = values[i];
    System.out.println(value);
}
```

We could rewrite the loop like this:

```
for (int value : values) {
    System.out.println(value);
}
```

This statement is called an enhanced for loop, also known as the "for each" loop. You can read the code as, "for each value in values". It's conventional to use plural nouns for array variables and singular nouns for element variables.

Notice how the single line for (int value : values) replaces the first two lines of the standard for loop. It hides the details of iterating each index of the array, and instead, focuses on the values themselves.

Using the enhanced for loop, we can write an accumulator very concisely:

```
int[] counts = new int[100];
for (int score : scores){
    counts[score]++;
}
```

Enhanced for loops often make the code more readable, especially for accumulating values. They are not helpful when you need to refer to the index, as in search operations:

```
for (double d : array){
    if (d == target){
        // array contains d, but we don't know where
    }
}
```

Tutorial 1: Looping Arrays

```java
/**
 * Accessing and Looping Arrays
 */
public class ArrayAccessLoops {

    public static void main(String[] args) {
        // Declare constant for array size
        final int SIZE = 4;
        int[] counts = new int[SIZE];

        // Traversal with a for loop
        System.out.println("\nFor loop with original empty array");
        for (int i = 0; i < 4; i++) {
            System.out.print(counts[i] + " ");
        }

        // Access individual array elements
        counts[0] = 7;
        counts[1] = counts[0] * 2;
        counts[2]++;
        counts[3] -= 60;

        // Traversal with a while loop
        System.out.println("\nWhile loop");
        // Counter for while loop
        int j = 0;
        while (j < 4) {
            System.out.print(counts[j] + " ");
            j++;
        }

        // Traversal with a for each loop
        System.out.println("\nFor each loop");
        // For each element in the array
        for (int count : counts) {
            System.out.print(count + " ");
        }

        // Multiply each element by 2
        System.out.println("\nFor loop multiply each element by 2");
        for (int i = 0; i < 4; i++) {
            int countsTwo = (counts[i] * 2);
            System.out.print(countsTwo + " ");
        }
    }
}
```

Example run:

```
For loop with original empty array
0 0 0 0
While loop
7 14 1 -60
For each loop
7 14 1 -60
For loop multiply each element by 2
14 28 2 -120
```

Tutorial 2: Arrays with Functions

```java
// Provides Array methods
import java.util.Arrays;

public class ArraysFunctions {
    public static void main(String[] args) {

        int[] array = { 1, 2, 3, 4 };

        // Displaying array with function with array argument
        System.out.println("Print Array using function with array argument");
        printArray(array);

        // Printing an array as an object
        // This shows the memory address of the object
        System.out.println("Print Array's memory address");
        System.out.println(array);

        // Printing with Arrays class
        System.out.println("Print Array using Arrays class method");
        System.out.println(Arrays.toString(array));

```

```
22          // Copying an array is done by copying the elements
23          // from one array to the next
24          double[] a = { 1.0, 2.0, 3.0 };
25          double[] b = new double[a.length];
26          System.out.println("Print copy b of Array a");
27          for (int i = 0; i < a.length; i++) {
28              b[i] = a[i];
29              System.out.print(b[i] + " ");
30          }
31
32          System.out.println("\nPrint copy c of Array a");
33          // Copying with Arrays class method
34          double[] c = Arrays.copyOf(a, a.length);
35          for (double i : c) {
36              System.out.print(i + " ");
37          }
38
39
40          // Search an array using a method with parameters
41          int index = search(a, 2.0);
42          System.out.println("\nindex = " + index);
43
44          // reduce
45          double total = sum(a);
46          System.out.println("total = " + total);
47      }
```

```
49      /**
50       * Function to print the elements of an array
51       */
52      public static void printArray(int[] a) {
53          System.out.print("{" + a[0]);
54          for (int i = 1; i < a.length; i++) {
55              System.out.print(", " + a[i]);
56          }
57          System.out.println("}");
58      }
59
60      /**
61       * Returns the index of the target in the array, or -1 if not found.
62       */
63      public static int search(double[] array, double target) {
64          for (int i = 0; i < array.length; i++) {
65              if (array[i] == target) {
66                  return i;
67              }
68          }
69          return -1; // not found
70      }
71
72      /**
73       * Returns the total of the elements in an array
74       */
75      public static double sum(double[] array) {
76          double total = 0.0;
77          for (int i = 0; i < array.length; i++) {
78              total += array[i];
79          }
80          return total;
81      }
82  }
```

Example run:

```
Print Array using function with array argument
{1, 2, 3, 4}
Print Array's memory address
[I@85ede7b
Print Array using Arrays class method
[1, 2, 3, 4]
Print copy b of Array a
1.0 2.0 3.0
Print copy c of Array a
1.0 2.0 3.0
index = 1
total = 6.0
```

ArrayLists

ArrayLists are **dynamically sized.** Developers don't need to specify the capacity or the maximum size when declaring ArrayLists. As elements are added and removed, it **grows or shrinks** its size automatically.

The previous Array operations hold for ArrayLists. Here are some additional methods as the ArrayList is mutable.

ArrayList Operations

- **add(element)** to add a new element to the end of this list or **add(index, element)** to add element at the specified index.
- **addAll(ArrayList)** to append another ArrayList) to it.
- **set(index, element)** replaces the element at the specified position in this list with the specified element.
- **get(index)** to get an element at the index.
- **isEmpty()** to check if there are any elements in this list. Returns true if this list contains no elements.
- **size()** returns the number of elements in this list.
- **clear()** to delete all elements. You can also use removeAll() to achieve the same, but it is slower than clear().

- **remove(index)** removes the element at the index in this list. Shifts any subsequent elements to the left (subtracts one from their indices).
- **remove(element)** removes the first occurrence of the specified element from this list.
- **removeIf(predicate)** removes element from the ArrayList that satisfy the predicate argument e.g. removeIf(s -> s.length == 3)
- **contains(element)**, **indexOf(element)** and **lastIndexOf(element)** methods are used for searching for elements in ArrayList.
- **listIterator()** to get an ordered list iterator over the elements in this list.
- **toArray()** converts this ArrayList into an array.

ArrayList

Let's create an ArrayList of strings.

```
ArrayList<String> stringList=new ArrayList<String>();
```

This statement creates an ArrayList with the name aList with type "String". The type determines which type of elements the list will have. Since this list is of "String" type, the elements that are going to be added to this list will be of type "String".

Here we create an ArrayList that accepts int elements.

```
ArrayList<Integer> intList=new ArrayList<Integer>();
```

Add Elements to an ArrayList

We add elements to an ArrayList by using **add() method**, this method has couple of variations, which we can use based on the requirement.

For example: If we want to add the element at the end of the List then simply do it like this:

```
alist.add("Steve"); //This will add "Steve" at the end of List
```

To add the element at the specified location in ArrayList, we can specify the index in the add method like this:

```
alist.add(3, "Steve"); //This will add "Steve" at the fourth position
```

A complete example.

```
import java.util.ArrayList;

public class ArrayListAddElement {

    public static void main(String args[]) {
        ArrayList<String> alist = new ArrayList<String>();
        alist.add("Steve");
        alist.add("Tim");
        alist.add("Lucy");
        alist.add("Pat");
        alist.add("Angela");
        alist.add("Tom");

        // displaying elements
        System.out.println(alist);

        // Adding "Steve" at the fourth position
        alist.add(3, "Steve");

        // displaying elements
        System.out.println(alist);
    }
}
```

Example run:

```
[Steve, Tim, Lucy, Pat, Angela, Tom]
[Steve, Tim, Lucy, Steve, Pat, Angela, Tom]
```

Change an Element in an ArrayList

We use the set method to change an element in ArrayList. We provide the index and new element, this method then updates the element present at the given index with the new given element. In the following example, we have given the index as 0 and new element as "Lucy" in the set() method, so the method updated the element present at the index 0 (which is the first element "Jim" in this example) with the new String element "Lucy", which we can see in the output.

```
import java.util.ArrayList;
public class ArrayListModifyElement {
   public static void main(String[] args) {
      ArrayList<String> names = new ArrayList<String>();
      names.add("Jim");
      names.add("Jack");
      names.add("Ajeet");
      names.add("Chaitanya");
      // Modify the first element
      names.set(0, "Lucy");
      System.out.println(names);
   }
}
```

Example run:

```
[Lucy, Jack, Ajeet, Chaitanya]
```

Remove Elements from an ArrayList

We use the remove() method to remove elements from an ArrayList.

```
import java.util.*;

public class ArrayListRemoveElement {

    public static void main(String args[]) {
        ArrayList<String> alist = new ArrayList<String>();
        alist.add("Steve");
        alist.add("Tim");
        alist.add("Lucy");
        alist.add("Pat");
        alist.add("Angela");
        alist.add("Tom");

        // Display elements
        System.out.println(alist);

        // Remove "Steve" and "Angela"
        alist.remove("Steve");
        alist.remove("Angela");

        // Displaying elements
        System.out.println(alist);

        // Remove 3rd element
        alist.remove(2);

        // Display elements
        System.out.println(alist);
    }
}
```

Example run:

```
[Steve, Tim, Lucy, Pat, Angela, Tom]
[Tim, Lucy, Pat, Tom]
[Tim, Lucy, Tom]
```

Iterating Through an ArrayList

In the above examples, we have displayed the ArrayList elements just by referring the ArrayList instance, which is not the right way to displays the elements. The correct way of displaying the elements is by using an advanced for loop like this.

We can use size() method of ArrayList to find the number of elements in an ArrayList.

```
import java.util.*;

public class ArrayListIteration {

    public static void main(String args[]) {
        ArrayList<String> alist = new ArrayList<String>();
        alist.add("Gregor Clegane");
        alist.add("Khal Drogo");
        alist.add("Cersei Lannister");
        alist.add("Sandor Clegane");
        alist.add("Tyrion Lannister");

        // Use size method to find the length of the arraylist
        System.out.println("This ArrayList has " + alist.size() + "
elements.");

        // Iterating through an ArrayList
        for (String str : alist)
            System.out.println(str);
    }
}
```

Example run:

```
This ArrayList has 5 elements.
Gregor Clegane
Khal Drogo
Cersei Lannister
Sandor Clegane
Tyrion Lannister
```

ArrayList Example in Java

This example demonstrates how to create, initialize, add and remove elements from ArrayList. In this example we have an ArrayList of type “String”.

```
import java.util.ArrayList;

public class ArrayListExample {

    public static void main(String args[]) {
        /*
         * Creating ArrayList of type "String" which means
         * we can only add "String" elements
         */
        ArrayList<String> obj = new ArrayList<String>();

        /* This is how we add elements to an ArrayList */
        obj.add("Ajeet");
        obj.add("Harry");
        obj.add("Chaitanya");
        obj.add("Steve");
        obj.add("Anuj");

        // Displaying elements
        System.out.println("Original ArrayList:");
        for (String str : obj)
            System.out.println(str);

        /*
         * Add element at the given index
         * obj.add(0, "Rahul") - Adding element "Rahul" at first position
         * obj.add(1, "Justin") - Adding element "Justin" at second position
         */
        obj.add(0, "Rahul");
        obj.add(1, "Justin");

        // Displaying elements
        System.out.println("ArrayList after add operation:");
        for (String str : obj)
            System.out.println(str);

        // Remove elements from ArrayList like this
        obj.remove("Chaitanya"); // Removes "Chaitanya" from ArrayList
        obj.remove("Harry"); // Removes "Harry" from ArrayList

        // Displaying elements
```

```
        System.out.println("ArrayList after remove operation:");
        for (String str : obj)
            System.out.println(str);

        // Remove element from the specified index
        obj.remove(1); // Removes Second element from the List

        // Displaying elements
        System.out.println("Final ArrayList:");
        for (String str : obj)
            System.out.println(str);
    }
}
```

Example run:

```
Ajeet
Harry
Chaitanya
Steve
Anuj
ArrayList after add operation:
Rahul
Justin
Ajeet
Harry
Chaitanya
Steve
Anuj
ArrayList after remove operation:
Rahul
Justin
Ajeet
Steve
Anuj
Final ArrayList:
Rahul
Ajeet
Steve
Anuj
```

Tutorial 3: ArrayLists

```java
import java.util.ArrayList;

public class ArrayListDemo {

   public static void main(String args[]) {
      // Create an empty array list to store integers
      ArrayList<Integer> numbers = new ArrayList<>();

      System.out.println("Initial size of Numbers ArrayList: " + numbers.size());

      // Add elements to the array list
      numbers.add(3);
      numbers.add(4);
      numbers.add(10);
      numbers.add(56);
      numbers.add(129);
      numbers.add(34);
      numbers.add(1, 12);
      System.out.println("Size of Numbers after additions: " + numbers.size());

      // display the array list
      System.out.println("Contents of Numbers: " + numbers);

      // Remove elements from the array list
      // Remove element by value
      numbers.remove(Integer.valueOf(2));
      // Remove element at specific index
      numbers.remove(2);

      System.out.println("Size of Numbers after deletions: " + numbers.size());
      System.out.println("Contents of Numbers: " + numbers);
   }
}
```

Example run:

```
Initial size of Numbers ArrayList: 0
Size of Numbers after additions: 7
Contents of Numbers: [3, 12, 4, 10, 56, 129, 34]
Size of Numbers after deletions: 6
Contents of Numbers: [3, 12, 10, 56, 129, 34]
```

Assignment Submission

1. Attach the program files.

2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.